Projet d'Intégration Continue: Refactoring et Pipeline DevOps

Contexte du Projet
Les étudiants doivent transformer une application Java spaghetti en une base de code maintenable, puis implémenter un pipeline CI/CD complet avec analyse de qualité.

Objectifs Pédagogiques

1. Comprendre les anti-patterns et leur impact
2. Maîtriser 3 design patterns fondamentaux
3. Implémenter un pipeline CI/CD professionnel
4. Générer des rapports techniques automatisés

Code Spaghetti de Départ ; Fichier: `SpaghettiFinanceApp.java`

Fichier: `pom.xml` (version initiale minimaliste)

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.university</groupId>
    <artifactId>spaghetti-finance</artifactId>
    <version>1.0.0</version>
    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Tâches du Projet

Phase 1: Analyse du Code Spaghetti

1. Identifier les anti-patterns présents
2. Lister les violations des principes SOLID
3. Proposer une architecture modulaire

Phase 2: Refactoring avec Design Patterns

Pattern 1: STRATEGY

- Pour les différents types de transactions
- Créer une interface **`TransactionStrategy`**

Pattern 2: FACTORY

- Pour la création d'utilisateurs et comptes
- Implémenter `AccountFactory` et `UserFactory`

Pattern 3: OBSERVER

- Pour la notification des transactions
- Système d'audit et notifications

Pattern Bonus (Optionnel): SINGLETON

- Pour le gestionnaire de configuration

Phase 3: Tests Unitaires avec JUnit

1. Couverture de test > 80%
2. Tests pour chaque pattern implémenté
3. Tests d'intégration des modules

Phase 4: Pipeline CI/CD avec Jenkins
Fichier: `Jenkinsfile`

Phase 5: Analyse de Qualité avec SonarQube

1. Configuration SonarQube
2. Métriques à surveiller :
    - Technical Debt
    - Code Smells
    - Coverage
    - Vulnerabilities

Structure du Projet Final

```
refactored-finance-app/
├── src/main/java/com/university/finance/
│   ├── pattern/strategy/
│   │   ├── TransactionStrategy.java
│   │   ├── DepositStrategy.java
│   │   ├── WithdrawStrategy.java
│   │   └── TransferStrategy.java
│   ├── pattern/factory/
│   │   ├── AccountFactory.java
│   │   └── UserFactory.java
│   ├── pattern/observer/
│   │   ├── TransactionObserver.java
│   │   ├── AuditLogger.java
│   │   └── NotificationService.java
│   ├── model/
│   │   ├── Account.java
│   │   ├── User.java
│   │   └── Transaction.java
│   ├── service/
│   │   ├── BankingService.java
│   │   └── TransactionService.java
│   └── MainApp.java
├── src/test/java/
│   └── (tests unitaires et d'intégration)
├── pom.xml
├── Jenkinsfile
├── sonar-project.properties
└── README.md
```

Livrables Attendus

Livrable 1: Code Refactoré

- Application modulaire avec 3 patterns
- Code source commenté et documenté
- Respect des conventions Java

Livrable 2: Tests Unitaires

- Minimum 20 tests unitaires
- Couverture > 80%
- Rapports JUnit générés

Livrable 3: Pipeline CI/CD

- Jenkinsfile fonctionnel
- Build automatique sur Git push
- Notifications par email

Livrable 4: Rapports de Qualité

- Rapport SonarQube complet
- Analyse des métriques
- Plan d'amélioration

Livrable 5: Documentation

- Diagrammes UML
- Guide d'installation
- Présentation des patterns

Configuration Technique Requise

Fichier: `sonar-project.properties`

```
sonar.projectKey=finance-refactoring
sonar.projectName=Finance App Refactoring
sonar.projectVersion=1.0
sonar.sources=src/main/java
sonar.tests=src/test/java
sonar.java.binaries=target/classes
sonar.junit.reportsPath=target/surefire-reports
sonar.coverage.jacoco.xmlReportPaths=target/site/jacoco/jacoco.xml
```

Critères d'Évaluation

Critère	Poids	Description
Qualité du refactoring	30%	Application correcte des patterns
Couverture de tests	20%	Tests complets et pertinents
Pipeline CI/CD	25%	Fonctionnalité et automatisation
Qualité du code	15%	Métriques SonarQube
Documentation	10%	Clarté et complétude

Bonus et Extensions Possibles (+5 points)

1. **Dockerisation** : Conteneuriser l'application
2. **Tests de Performance** : Avec JMeter
3. **Déploiement Automatique** : Sur serveur de test
4. **Monitoring** : Intégration Prometheus/Grafana

Améliorations Avancées:

- Pipeline Blue Ocean pour Jenkins
- Qualité du code avec Checkstyle/PMD
- Revue de code automatisée avec Gerrit
- Sécurité avec OWASP Dependency Check

Instructions Finales

1. **Forker** le dépôt spaghetti initial
2. **Travailler en équipes** de 3-4 personnes
3. **Utiliser Git Flow** pour la gestion des branches
4. **Commiter régulièrement** avec des messages clairs
5. **Documenter** chaque décision d'architecture
6. **Présenter** le projet en soutenance finale

Date de remise : 03/12/2025
Date de dépôt : 26/12/2025

Ce projet couvre l'ensemble du cycle de vie du développement logiciel moderne, de la dette technique à l'intégration continue professionnelle. Les étudiants développeront des compétences techniques précieuses tout en comprenant l'importance de la qualité du code dans un contexte DevOps.